AGENTI COME PATTERN

AGENT-ORIENTED COMPUTING SU LINGUAGGI OBJECT-ORIENTED

Alessandro Ricci
a.ricci@unibo.it
DEIS, Università di Bologna
Sede di Cesena

Agenti come pattern

- Obiettivo: realizzare l'astrazione di agente in termini di pattern su linguaggi mainstream
  - Object-oriented (es: Java)
  - Liguaggi dichiarativi (es: Prolog)
  - ...
- Letteratura abbondante
  - in particolare: lavoro della Kendall (*)

(*) "A Java Application Framework for Agent Based Systems" - Kendall, Krishna, Pathak, Suresh (RMIT)

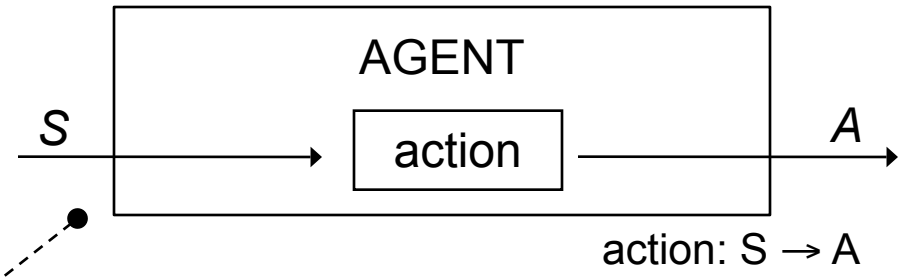
Agent abstract loop semplificato

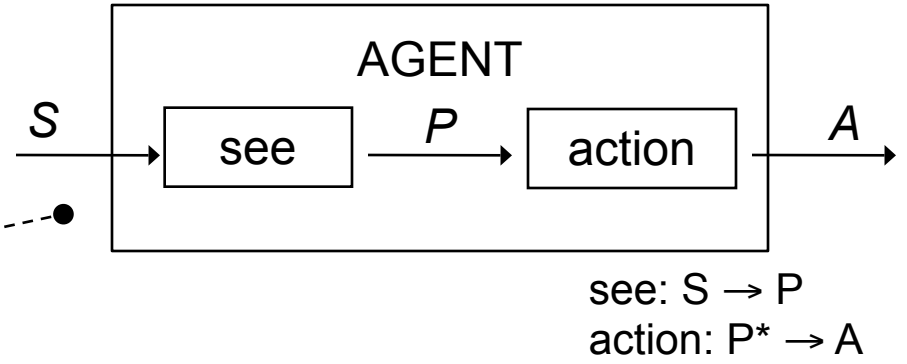
```
while (living) {
  percept();
  compute();.[reason,plan,...]
  act();
}
```

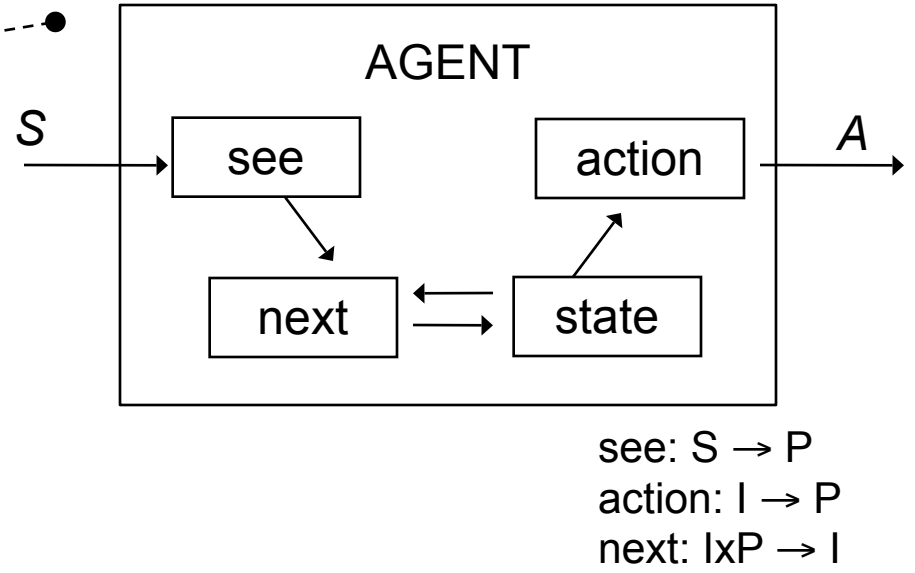
seleziona l'azione da compiere, secondo un determinato criterio

Architettura astratta di base

- Puramente reattivi
- Percezione
- Con stato







S:set of input (environment partial state)
A:set of actions
P:set of perceptions
I: set of agent internal states

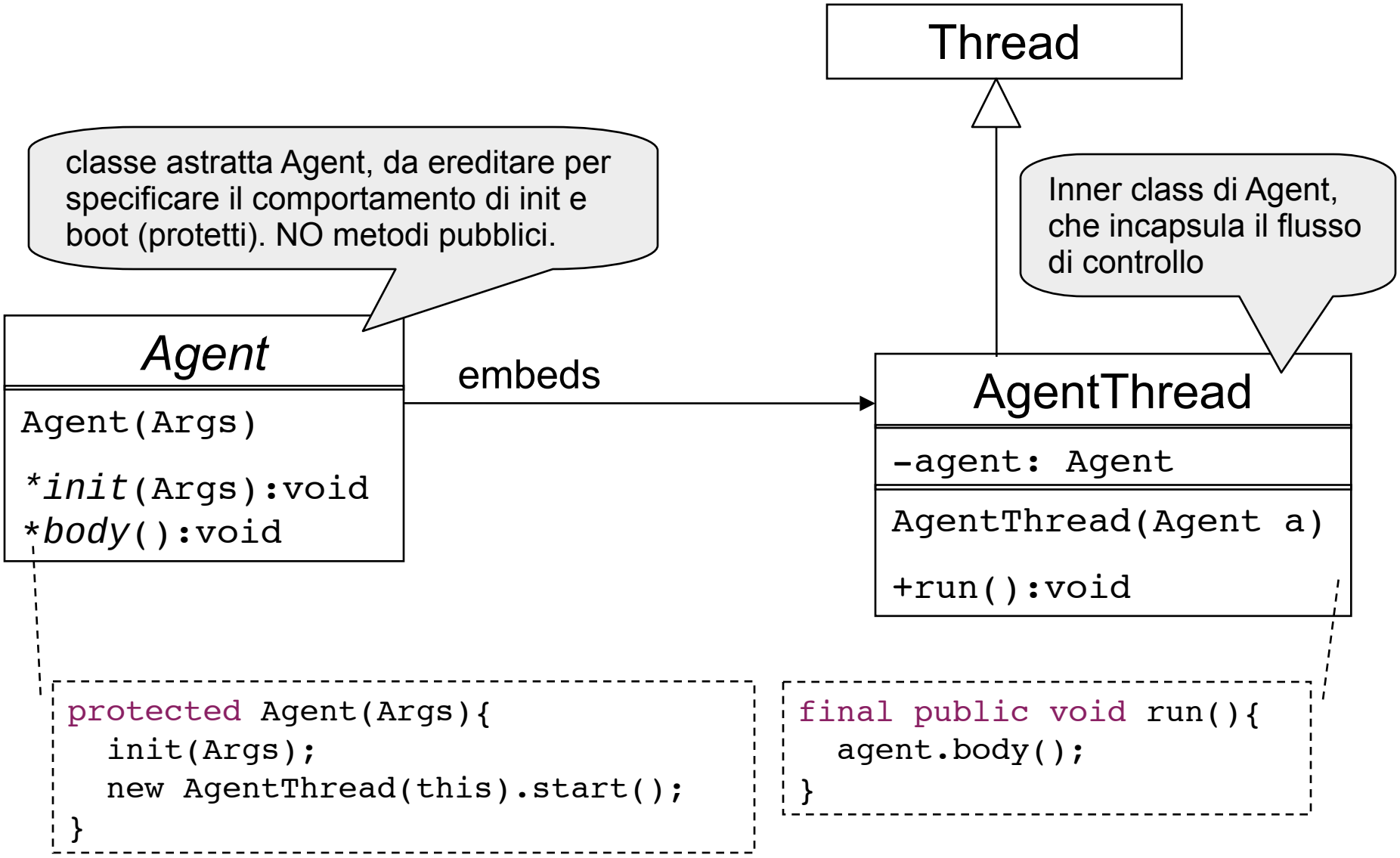
Architetture di riferimento

- Architetture reattive / behaviour-based
  - Automi a stati finiti
  - Architetture sussunzione
  - Situated automata
- Architetture logiche
  - Rules-based
  - Architetture deliberative / cognitive
    - BDI (Belief Desire Intention)
- Architetture ibride
  - A livelli orizzontali e verticali

Esempio semplice su linguaggi OO

- Agente non ha interfaccia à-la OO
  - Nessun metodo pubblico
    - “public methods considered harmful”
- Ogni interazione con l’environment deve avvenire mediante le astrazioni/meccanismi fornite dal modello di interazione
  - può sfruttare e interagire con oggetti / servizi direttamente (API Object-Oriented), se queste sono viste come sue risorse private
- Incapsulamento flusso di controllo

Agente di base: bozza di pattern



Definizione di un agente: esempio

```
public class MyAgent extends Agent {

  private long stateVar;

  protected void init(AgentParams p){
    System.out.println("initialisation...");
    stateVar = 0;
  }

  protected void body(){
    while (true){
      System.out.println("I'm still alive after "+stateVar+"cycles");
      stateVar++;
      try {
        sleep(1000);
      } catch (Exception ex){}
    }
  }
}
```

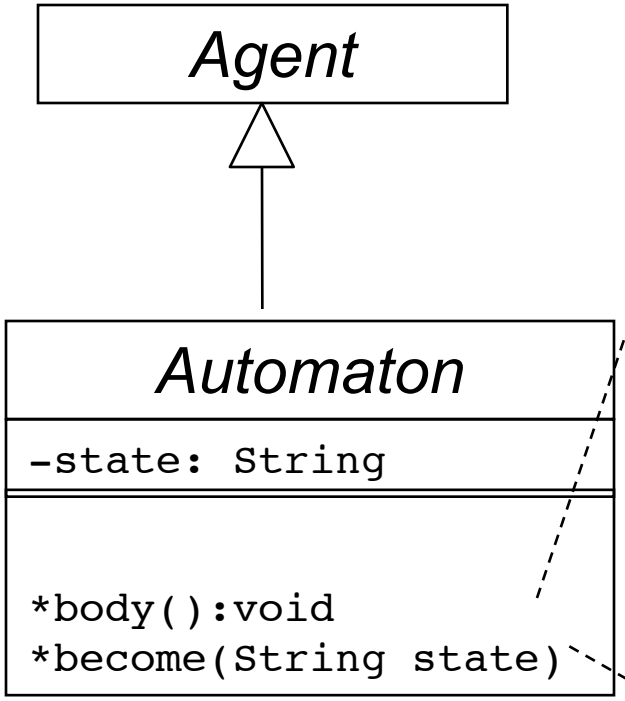
Esempio più articolato: automa a stati

- Agente come *automa a stati*
  - il comportamento viene definito come insieme di stati, fornendo la possibilità di specificare la transizione di stato
  - alla nascita l'agente è nello stato boot
  - transitare nello stato end = morte
- In generale 2 possibili design
  - Reflection pattern
  - State-Behaviour pattern

Reflection pattern

- Stati/Comportamenti definiti da metodi protetti della classe agent
  - alcuni di default: `idle, boot, end`
  - nome del metodo come identificatore dello stato
- Meccanismo per specificare transizione di stato
  - metodo `become(String stateName)`
- Si può ottenere estendendo il pattern di base, specializzando il comportamento di `body`

Automi con reflection: bozza di pattern




```
final protected void body(){
  while (true){
    if (!state.equals("end")){
      Method m  =  this.getClass().
         getDeclaredMethod(state,null);
      m.setAccessible(true);
      m.invoke(this,null);
    } else {
      end();
      break;
    }
  }
}
```

```
protected void become(String s){
  if (!state.equals("end")){
    state=s;
  }
}
```

Definizione di un automa: esempio

```
public class MyAutomaton extends Agent {

  private long stateVar;

  protected void boot(){
    stateVar = 0;
    become(“myBehaviourA”);
  }

  protected void myBehaviourA(){
    stateVar++;
    if (stateVar>100){
      become(“myBehaviourB”);
    } else {
      become(“myBehaviourA”);
    }
  }

 protected void myBehaviourB(){
    stateVar--;
    if (stateVar==0){
      become(“myBehaviourA”);
    } else {
      become(“myBehaviourB”);
    }
  }
}
```

State-Behaviour pattern

- Stati / comportamenti rappresentati direttamente come classi
  - riferimento al pattern State (*)

(*) “Design Pattern” -Gamma et al. - Addison Wesley

Esempio: piattaforma JADE

```
import jade.core.Agent;
import jade.core.behaviours.*;


public class MyAgent extends Agent     {
  protected void setup()          {
    addBehaviour( new MyBehaviour( this ) );
  }
  class MyBehaviour extends SimpleBehaviour{
    public myBehaviour(Agent a) {
      super(a);
    }
    public void action() {
     //...this is where the real programming goes !!
    }
    private boolean finished = false;
    public boolean done() {
      return finished;
    }
  } // ------------ End myBehaviour
}//end class myAgent
```

Esempio di estensioni: JACK

- JACK è una piattaforma per lo sviluppo di agenti basati su una *estensione* del linguaggio Java
  - aggiunti costrutti per specificare aspetti di tipo mentalistico
  - Modello di riferimento: BDI
    - desires → obiettivi
    - intentions → piani per raggiungere obiettivi
    - beliefs → conoscenza del mondo

Costrutti nuovi JACK

- Agent
  - comportamento dell'agente
  - include capabilities, eventi/messaggi, piani
- Capability
  - incapsula piani, beliefset, eventi per il riuso
- BeliefSet
  - conoscenza dell'agente in un modello relazionale
- View
  - query sulla conoscenza
- Event
  - eventi significativi per l'agente, a cui deve reagire con opportune azioni
- Plan
  - istruzioni per raggiungere determinati obiettivi

Aspetto di un agente JACK

```
event MyEvent extends
        BDIGoalEvent {
  String text; // For example.
  #posted as
       myPostingMethod(String s){
    text = s;
  }
}
```

```
plan MyPlan extends Plan {
  #handles event MyEvent me;
  body(){
    System.out.println(me.text);
  }
}
```

```
agent MyAgent extends Agent {

  #handles event MyEvent;
  #uses plan MyPlan;
  #posts event MyEvent myEventRef;

  MyAgent(String name){
    super(name);
  }
  public void method1(String exampleMessage){
    postEvent(myEventRef.myPostingMethod(exampleMessage));
  }
}
```

```
public class Program
{
  public static void main(String args[])
{
    MyAgent agent1 = new
                          MyAgent
("agent1");
    agent1.method1("data to be posted");
  }
}
```